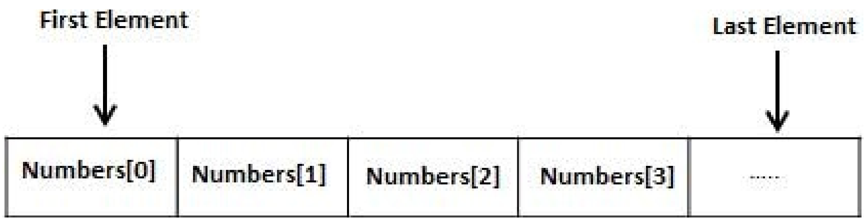
VB.NET - ARRAYS

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30)    ' an array of 31 elements
Dim strData(20) As String ' an array of 21 strings
Dim twoDarray(10, 20) As Integer 'a two dimensional array of integers
Dim ranges(10, 100)  'a two dimensional array
```

You can also initialize the array elements while declaring the array. For example,

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim names() As String = {"Karthik", "Sandhya", _
"Shivangi", "Ashwitha", "Somnath"}
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

```
Module arrayApl
   Sub Main()
      Dim n(10) As Integer  ' n is an array of 10 integers '
      Dim i, j As Integer
      ' initialize elements of array n '
      For i = 0 To 10
          n(i) = i + 100 ' set element at location i to i + 100
      Next i
      ' output each array element's value '
      For j = 0 To 10
          Console.WriteLine("Element({0}) = {1}", j, n(j))
      Next j
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces following result:

```
Element(0) = 100
Element(1) = 101
Element(2) = 102
Element(3) = 103
Element(4) = 104
Element(5) = 105
Element(6) = 106
Element(7) = 107
Element(8) = 108
Element(9) = 109
Element(10) = 110
```

Dynamic Arrays

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the **ReDim** statement.

Syntax for ReDim statement:

```
ReDim [Preserve] arrayname(subscripts)
```

Where,

- The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
- **arrayname** is the name of the array to re-dimension
- **subscripts** specifies the new dimension.

```
Module arrayApl
   Sub Main()
      Dim marks() As Integer
      ReDim marks(2)
      marks(0) = 85
      marks(1) = 75
      marks(2) = 90
      ReDim Preserve marks(10)
      marks(3) = 80
      marks(4) = 76
      marks(5) = 92
      marks(6) = 99
      marks(7) = 79
      marks(8) = 75
      For i = 0 To 10
          Console.WriteLine(i & vbTab & marks(i))
      Next i
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces following result:

```
0 85
1 75
2 90
3 80
4 76
5 92
6 99
7 79
8 75
9 0
10 0
```

Multi-Dimensional Arrays

VB.Net allows multidimensional arrays. Multi-dimensional arrays are also called rectangular array.

You can declare a 2 dimensional array of strings as:

```
Dim twoDStringArray(10, 20) As String
```

or, a three dimensional array of Integer variables:

```
Dim threeDIntArray(10, 10, 10) As Integer
```

The following program demonstrates creating and using a two dimensional array:

```
Module arrayApl
   Sub Main()
      ' an array with 5 rows and 2 columns
      Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
      Dim i, j As Integer
      ' output each array element's value '
      For i = 0 To 4
          For j = 0 To 1
              Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
          Next j
      Next i
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces following result:

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

Jagged Array

A Jagged array is an array of arrays. The follwoing code shows declaring a jagged array named *scores* of Integers:

```
Dim scores As Integer()() = New Integer(5)(){}
```

The following example illustrates using a jagged array:

```
Module arrayApl
   Sub Main()
      'a jagged array of 5 array of integers
      Dim a As Integer()() = New Integer(4)() {}
      a(0) = New Integer() {0, 0}
      a(1) = New Integer() {1, 2}
      a(2) = New Integer() {2, 4}
      a(3) = New Integer() {3, 6}
      a(4) = New Integer() {4, 8}
      Dim i, j As Integer
      ' output each array element's value
      For i = 0 To 4
          For j = 0 To 1
              Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i)(j))
          Next j
      Next i
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces following result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

The Array Class

The Array class is the base class for all the arrays in VB.Net. It is defined in the System namespace. The Array class provides various properties and methods to work with arrays.

Properties of the Array Class

The following table provides some of the most commonly used **properties** of the **Array** class:

S.N	Property Name & Description
1	**IsFixedSize** Gets a value indicating whether the Array has a fixed size.
2	**IsReadOnly** Gets a value indicating whether the Array is read-only.
3	**Length** Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	**LongLength** Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	**Rank** Gets the rank (number of dimensions) of the Array.

Methods of the Array Class

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
1	**Public Shared Sub Clear (array As Array, index As Integer, length As Integer)** Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	**Public Shared Sub Copy (sourceArray As Array, destinationArray As Array, length As Integer)** Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
3	**Public Sub CopyTo (array As Array, index As Integer)** Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer.

4	**Public Function GetLength (dimension As Integer) As Integer** Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
5	**Public Function GetLongLength (dimension As Integer) As Long** Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
6	**Public Function GetLowerBound (dimension As Integer) As Integer** Gets the lower bound of the specified dimension in the Array.
7	**Public Function GetType As Type** Gets the Type of the current instance. (Inherited from Object.)
8	**Public Function GetUpperBound (dimension As Integer) As Integer** Gets the upper bound of the specified dimension in the Array.
9	**Public Function GetValue (index As Integer) As Object** Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
10	**Public Shared Function IndexOf (array As Array, value As Object) As Integer** Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
11	**Public Shared Sub Reverse (array As Array)** Reverses the sequence of the elements in the entire one-dimensional Array.
12	**Public Sub SetValue (value As Object, index As Integer)** Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
13	**Public Shared Sub Sort (array As Array)** Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.
14	**Public Overridable Function ToString As String** eturns a string that represents the current object. (Inherited from Object.)

For complete list of Array class properties and methods, please consult Microsoft documentation.

Example

The following program demonstrates use of some of the methods of the Array class:

```
Module arrayApl
   Sub Main()
      Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
      Dim temp As Integer() = list
      Dim i As Integer
      Console.Write("Original Array: ")
      For Each i In list
         Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      ' reverse the array
      Array.Reverse(temp)
      Console.Write("Reversed Array: ")
      For Each i In temp
```

```
            Console.Write("{0} ", i)
        Next i
        Console.WriteLine()
        'sort the array
        Array.Sort(list)
        Console.Write("Sorted Array: ")
        For Each i In list
            Console.Write("{0} ", i)
        Next i
        Console.WriteLine()
        Console.ReadKey()
    End Sub
End Module
```

When the above code is compiled and executed, it produces following result:

```
Original Array: 34 72 13 44 25 30 10
Reversed Array: 10 30 25 44 13 72 34
Sorted Array: 10 13 25 30 34 44 72
```